Towards secure SDN policy management

Nicolae Paladi
SICS Swedish ICT
nicolae@sics.se

***Abstract*—Software-Defined Networking (SDN) has emerged as a novel network architectural model that facilitates management of large-scale networks, enables efficient network virtualization and scalable network multi-tenancy. Centralized network controllers, an important component in the SDN paradigm, deploy on the data plane devices network policies from several independent sources, defined based on a global network view. While this approach allows to efficiently manage network connectivity and reduce the time and cost of deploying new configurations, it also increases the risk for errors – either introduced by accident, through a combination with previous policies, or by a motivated adversary. In this position paper we review the state of the art for network policy verification for SDN deployments, identify existing challenges and outline a secure framework for network policy management in SDN deployments. Combined with existing work on cloud platform and storage security, this will contribute towards creating secure and trusted cloud deployments.**

I. Introduction

Rapid proliferation of web services and cloud computing has triggered a remarkable increase in the number of data centers which rely on virtualization to allow infrastructure providers multiplex physical resources and provide complete platform and network resources to multiple tenants. Such platform resources are used as building blocks for increasingly complex network deployments necessary to support a multitude of web services. Along with enabling more advanced functionality, such complex network deployments also introduce intricate errors which command time and significant amounts of resources for investigation and resolution [1], [2], [3], [4]. As a result, cloud tenants may suffer business losses due to service outage, waste cloud resources and even lose valuable data.

The software-defined network model (SDN) emerged and rapidly evolved in response to the increasing complexity of network deployments, allowing to facilitate operation and management of cloud-grade networks [5], [6], [7], [8], [9], [10], [11]. The operational advantages of the SDN model lead to its increasing adoption in enterprise-grade network deployments on a global scale [12].

A conceptual model of the SDN architecture is depicted in Figure 1 and described below based on the SDN architectural model presented in [11].

- The *data plane* contains both hardware and software routing equipment. This component implements the routing policies that satisfy the goals of the network administrator. It lacks decision logic and is optimized for forwarding speed. Packets that do not match any policy are either discarded or communicated to the control plane through the southbound API.

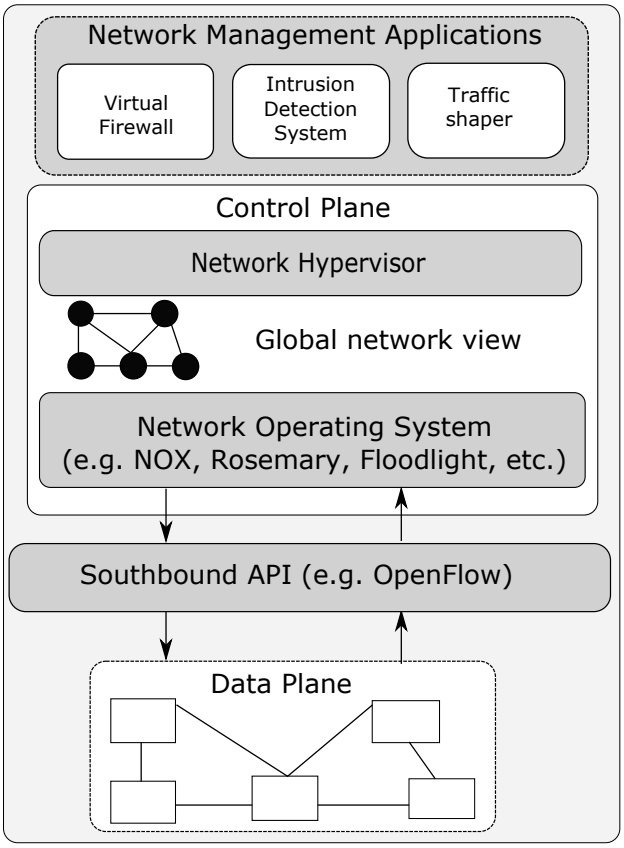

Fig. 1. High-level overview of the SDN architectural model

- *Southbound API* is a vendor-agnostic set of instructions implemented by the routing equipment on the data plane. It allows bi-directional communication between the data and the control planes.
- *Control plane* is a logically distributed abstraction layer that transforms high-level network operator goals into discrete routing policies based on a global network view. It contains a distributed *network operating system*, which builds and maintains the global network view as well as communicates with the equipment on the data plane. The control plane also includes the *network hypervisor*, which multiplexes the available network resources among multiple users with distinct virtual network topologies.
- *Management applications* are used by network administrators to express their network configuration goals using a set of high-level comments. They could also include software-based network management components such as

firewalls, intrusion detection systems, traffic shapers, etc.

In the process of operating the SDN deployment, the logically centralized control plane constructs a global view of the network components in its domain. This allows network management programs to rely on simpler graph processing algorithms to compute the shortest paths and to operate with higher-level abstractions, network operation is steered through network policies originating from three sources:

- High-level goals expressed by the network administrator and compiled into low-level configuration instructions for data-plane devices.
- Network management applications implemented as software components – included in the umbrella term *network function virtualization* (NFV) – which issue policies to implement their network functionality, e.g. as firewalls, traffic shapers, load balancers, instrusion detection devices and other functionality traditionally implemented in network middleboxes.
- Network operating systems (e.g. as described in [13], [14], [15], [16]), which may independently generate network policies in order to ensure network liveness properties in the face of unexpected events (e.g. severe traffic anomalies or a DDoS attack on a subset of network components).

The constant stream of policies from the sources described above – implemented by the network controller in a centralized manner throughout the deployment – leads to a continuous evolution of the network state. This situation leads to a new type of network configuration problems, since such network policies may have competing or conflicting effects on the data routing. In a security context, such network policy conflicts can lead to data leaks and isolation breaches in multi-tenant SDN environments. Thus, new algorithms are required for both static and run-time verification of network configuration against policy invariants.

Similar problems – such as static network reachability analysis or analysis of security policies for some protocols – have been studied before [17], [18], triggered by the increasing complexity of network deployments. However, such approaches are not fully applicable to large-scale SDN deployments due to their limited scalability and focus on low-level, protocol-specific details. More recent work [19], [20], [21], [22], [23] focused on SDN deployments and described targeted approaches for policy and topology verification against invariants defined by network administrators.

Finally, [24] proposed a security-enforcement kernel for network controllers that addresses challenges such as enabling application co-existence, detecting and resolving conflicts between incoming candidate flow rules and existing flow rules, creating an application permission model introducing application accountability and enforcing privilege separation.

Along with the rapid evolution of the state of the art in network policy verification and enforcement for SDN deployments and rapid progress towards mature and secure SDN controllers, a range of challenging problems and gaps continue to persist. Examples of such challenges are verifying liveness network properties (currently ignored in favor or safety properties), verifying policy composition for out-of-order rule installations, developing a model for non-interference among co-resident applications, as well as creating a sandboxing model for NFV applications interacting with the network operating system.

A. Contribution

In this paper, we outline the need for an integrated secure framework for creation, verification and enforcement of SDN policies. We review the current approaches for policy verification in SDN deployments, discuss their strengths and identify limitations that remain to be addressed. Finally, we outline a framework for secure SDN policy creation, verification and enforcement with access control for sandboxing of SDN management applications. This preliminary design will serve as an input for a comprehensive security architecture for SDN infrastructure.

B. Organization

The rest of the paper is organized as follows. In Section II we discuss the state of the art in network state verification in SDN. Next, in Section III we summarize the most important limitations and gaps identified earlier. In Section IV we outline an architecture for secure SDN verification and enforcement. Finally, we conclude the paper in Section V.

II. State of the Art

In this section we describe the relevant related work to network policy verification for SDN deployments.

In [19], the authors of "Anteater" note that verifying network correctness in the data plane offers several advantages over verifying higher-level code such as configuration files and use static analysis of network device forwarding information bases to uncover problems in the data plane. Thus, Anteater converts the data plane information into boolean expressions, translates network invariants into instances of boolean satisfiability (SAT) problems, and checks the resultant SAT formulas using a SAT solver to check for violations of key network invariants, such as absence of routing loops and black holes. The approach in [19] is an "offline", static approach and aims to analyze a snapshot of the data plane. Hence, while it can be used for complementary static checks, it does not scale well to dynamic changes in the network and requires up to hundreds of seconds to check a single invariant.

Symbolic execution can catch bugs through exploration of all possible code paths but it is usually not tractable for large software. NICE [21] combines model checking, symbolic execution and search strategies to test unmodified controller programs by automatically generating carefully-crafted streams of packets under many possible event interleavings. In particular, it performs symbolic execution of OpenFlow applications and applies model checking to explore the state space of an entire OpenFlow network. NICE is a proactive approach that attempts to identify invalid system states by using a

simplified OpenFlow switch model. An implementation of NICE reported in [21] has found 11 bugs in the network topology which were caused by software components part of the SDN controller. Among the limitations of NICE is the implicit assumption that all packets reach the controller – thus excluding the cases of total or partial packet loss in a given equivalence class. Omitting these cases leads to a loss in both coverage and efficiency of the tool. Likewise, while focusing on symbolic execution of each event handler, NICE does not check the dependencies between handler invocations. A third shortcoming is the loss in coverage resulting from throttling the input that can modify the controller state. While this is done to prevent infinite execution trees – which arise in NICE when each state has at least one input that limits the controller state – it further distances the model tested by NICE from the complexity of an SDN deployment. Finally, while NICE is not suitable for run-time policy verification, it can be used for static verification of the network.

Another static verification approach is described in [22], which uses header space analysis to statically check network specifications and configurations to identify certain classes of failures, such as reachability failures, forwarding looks, traffic isolation and leakage problems. In the context of multi-tenant SDN networks, such static checking can detect issues that can lead to breach of inter-tenant isolation. The header space framework used in this approach is built on a geometric model, with packets modeled as points in a geometric space and network boxes as transfer functions on the same geometric space, such as the *header space*, *network space* and *slice network space*. The analysis uses set operations – such as intersection, union, complementation and difference – in order to determine the interaction between different spaces and identify potential issues. While the approach has been implemented and tested on an enterprise network, it maintains the limitations of [21] since the approach in [22] can only be applied to *snapshots* of the system. In addition, header space analysis identifies the existence of issues, but stops short of listing their potential causes, such a specific policy or set of policies applied earlier.

Some of the shortcomings of [21], [22] have been addressed in [20], which proposed an approach for verifying network-wide invariants in real-time. The authors aim to prevent network bugs by verifying each change in the networking behaviour before it takes effect and block changes that violate important invariants. To do this, they introduce "VerfiFlow", a layer between the SDN controller and network devices that checks for network-wide invariants dynamically as each forwarding rule is inserted, modified or deleted. To check network-wide invariants in large networks and in the presence of complex forwarding elements, "VeriFlow" introduces a series of adaptations. First, it monitors all network update events in the live network as they are generated by network control applications, devices, or the network operator. Second, it confines the verification activities exclusively to the parts of the network that may be influenced by the new update. Third, "VeriFlow" uses a custom algorithm for invariant checking rather than utilizing general-purpose SAT or binary decision diagram solvers. While "VeriFlow" represents an important step forward from the static network checking to real-time verification of network invariants well-suited for SDN deployments, it contains a series of limitations. First, "VeriFlow" does not distinguish between temporary invariant violations (e.g. occuring in an intermediate state while applying multiple policies), or "actual" violations (induced by mistake or by an adversary). This can lead to a large number of false positive violations identified. Second, "VeriFlow" does not consider SET action commands present in certain southbound APIs (e.g. OpenFlow) and empowers apps to instruct a switch to rewrite the header attributed of a matching flow. As a result, it fails to handle recursive flow rule logic chains established by virtual circuits [24].

A different, equally important aspect is handling enforcement of the rules and sandboxing of the network management applications.

In "Rosemary: A Robust, Secure, and High-Performance Network Operating System" [16], the authors present a network operating system focusing on network resilience in the presence of faulty or malicious applications by creating sandboxed environments for network applications. Sandboxing (also called micro-NOS) is achieved by launching each application in a separate process context with access to all of the libraries that the application requires. Each Micro-NOS also contains a resource monitor to supervise the applications and operates within the permission structure of Rosemary network operating system. In turn, the Rosemary network operating system is an application running on a commodity Linux distribution. The isolation offered by the micro-NOS allows to improve robustness, such that faulty or malicious applications are prevented from crashing the entire network operating system. Furthermore, the paper also aims to address security aspects in order to prevent malicious network applications from accessing internal data structures of other network applications. This is achieved by implementing an AppZone sandbox, where privileged system calls made by a network application are interposed and verified by the sandbox framework. To avoid the declared '20-30%' performance overhead, the authors recommend two optimisations called 'request pipelining' and 'trusted execution'. The latter in essence removes the sandbox isolation, allowing the application to run as a kernel process; however, this makes the security advantages of Rosemary less evident. Finally, the authors present the performance evaluation results, which show that the Rosemary network operating system performs roughly on par with the NOX [13] approach on a 1G link and can perform on par with NOX on a 10G link with the 'trusted execution' optimisation in place. This contribution highlights the need for improved security in the architectures of the proposed network operating system. However, one major drawback of the proposed approach is that it ignores distribution aspects, despite significant progresses in distributed network operating system design [14] and the demonstrated need for physical distribution of the control plane [12].

III. Challenges

In this section, based on the partial review of the state of the art in Section II, we outline the list of existing challenges for policy verification and enforcement in SDN deployments.

a) Verification of liveness properties: current approaches to SDN policy verification focus on safety properties, e.g. that a certain packet can *not* reach a certain endpoint. However, run-time verification of liveness properties – e.g. that intended packets can reach legitimate targets, even after new policies have been applied, must also be done, in order to avoid intended or even accidental denial-of-service.

b) Interleaved execution: similarly, current approaches assume that events are executed atomically, i.e. that policies are installed atomically throughout the deployment. Such approaches do not consider intrleavings and out-of-order rule installations, which could introduce new bugs or be exploited by adversaries. Thus, future solutions should also consider non-atomic policy installations.

c) Determining verification time: as noted in the discussion of "VeriFlow" [20] in Section II, premature verification of invariant violation – done over an intermediate state occurring while a set of policies is applied – may lead to multiple false positives. An alternative approach, such as the one adopted in [24] is to perform the verification *after* the entire set of policies has been applied; however, this may lead to temporary isolation violation between tenants. Future solutions must address this issue by eliminating the risk for network isolation breaches without triggering unsustainable volumes of false positives.

d) Enforcing non-interference: current approaches focusing on policy verification do not enforce non-interference between applications. In particular, the approach in [24] explicitly allows applications to override each other depending on their authorization. Future solutions must identify a suitable privilege model that would enforce non-interference between network applications without limiting network functionality.

e) Sandboxing network applications: while application sandboxing has not been the focus of earlier approaches, more recent contributions address the need to separate network applications from the SDN controller context ([16], [24]). However, to the best of our knowledge, there is currently no access control model designed to account for the specifics of network operating systems. Thus, future solutions should contain an access control model for network operating systems applicable to SDN deployments that would not induce prohibitive performance penalties as reported in [16].

IV. Secure SDN deployments

In this section, we briefly outline a secure framework for network policy management in SDN deployments. Combined with existing work on trusted virtual machine launch [25] and secure cloud storage [26], this will contribute towards creating a secure cloud architecture capable of providing to cloud tenants explicit security guarantees regarding platform integrity, storage confidentiality and network isolation, and even help address more complex problems such as cloud data geolocation [27], [28].

Figure 2 shows a high-level representation of an SDN deployment architecture, extended with several components – namely the *real-time policy checker* and the *offline policy checker*.

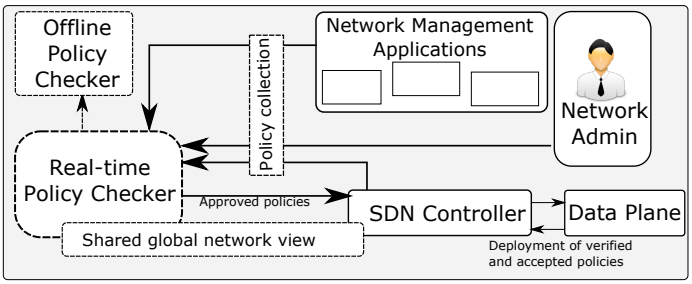

Fig. 2. High-level overview of the proposed SDN policy verification framework

Real-time policy checker: as the name suggests, the main task of this component is to conduct continuous real-time verification of the network policies (based on the approaches in [20], [24]) issued from the expected sources, as listed in Section I. In addition, this component is responsible for *origin tagging* of the policies, such that each issued policy can be traced to the issuing component. After a real-time verification, accepted policies are forwarded to the controller for deployment on the data plane devices. All information about the incoming, accepted and rejected policies (including their origin and execution context) is stored in a database for secondary, periodic analysis by the *offline policy checker* described below.

Offline policy checker: contrary to the *Real-time policy checker*, the *offline policy checker* conducts periodic, static verification of the policies based on snapshots of the network view, as described in [19], [21], [22]. While this can not prevent policy conflicts as they occur, the offline policy checker can perform periodic in-depth analysis of the network state against invariants. The aim of the offline policy checker is to verify complementary network properties – such as liveness and network reachability and tenant isolation – as well as monitor the deployed network applications for malicious activities. In this sense, the offline policy checker can perform a role similar to that of an intrusion detection system, but with a focus on detecting malicious policies.

Besides policy verification and enforcement, several additional aspects must be included in the framework in order to create a trusted and secure SDN infrastructure. Such aspects are: an access control model specifically designed to ensure efficient sandboxing of network applications and isolation between the different privilege levels; mechanisms for trusted deployment of the infrastructure and enrollment of the data plane devices; effective tenant isolation in multi-tenant SDN deployments and strong quota enforcement for reliable and fair distribution of resources among tenants.

V. Conclusion

In this position paper we have outlined the need for a comprehensive approach towards SDN policy management – including generation, verification and enforcement of network policies. Such an approach must take into account policies issued by the network operator, network applications and the network controller. An overview of the existing solutions shows that there has been steady progress over the years from static network analysis, to real-time policy verification to security-enforcement kernels and security-enhanced network controllers. Nevertheless, a set of challenges persists, such as verification of liveness properties during SDN policy verification, inclusion of interleaved policy execution, eliminating the risk for network isolation breaches without triggering unsustainable volumes of false positives, identifying a suitable privilege model that would enforce non-interference between network applications without limiting network functionality as well as defining an access control model for network operating systems applicable to SDN deployments with a minimal performance footprint.

References

[1] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *ACM SIGCOMM Computer Communication Review*, vol. 32, pp. 3–16, ACM, 2002.

[2] N. Feamster and H. Balakrishnan, "Detecting BGP configuration faults with static analysis," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pp. 43–56, USENIX Association, 2005.

[3] J. Wu, Z. M. Mao, J. Rexford, and J. Wang, "Finding a needle in a haystack: Pinpointing significant BGP routing changes in an IP network," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pp. 1–14, USENIX Association, 2005.

[4] Z. Yin, M. Caesar, and Y. Zhou, "Towards understanding bugs in open source router software," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 3, pp. 34–40, 2010.

[5] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and control element separation (ForCES) framework," tech. rep., RFC 3746, April, 2004.

[6] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.

[7] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "SANE: A Protection Architecture for Enterprise Networks.," in *Usenix Security*, 2006.

[8] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review*, vol. 37, pp. 1–12, ACM, 2007.

[9] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: dynamic access control for enterprise networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pp. 11–18, ACM, 2009.

[10] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson, "FRESCO: Modular Composable Security Services for Software-Defined Networks.," in *NDSS*, 2013.

[11] M. Casado, N. Foster, and A. Guha, "Abstractions for software-defined networks," *Communications of the ACM*, vol. 57, no. 10, pp. 86–95, 2014.

[12] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 3–14, ACM, 2013.

[13] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[14] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, *et al.*, "Onix: A Distributed Control Platform for Large-scale Production Networks.," in *OSDI*, vol. 10, pp. 1–6, 2010.

[15] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 121–126, ACM, 2012.

[16] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang, "Rosemary: A Robust, Secure, and High-Performance Network Operating System," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 78–89, ACM, 2014.

[17] G. G. Xie, J. Zhan, D. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, J. Rexford, *et al.*, "On static reachability analysis of IP networks," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3, pp. 2170–2183, IEEE, 2005.

[18] H. Hamed, E. Al-Shaer, and W. Marrero, "Modeling and verification of IPSec and VPN security policies," in *Network Protocols, 2005. ICNP 2005. 13th IEEE International Conference on*, pp. 10–pp, IEEE, 2005.

[19] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. Godfrey, and S. T. King, "Debugging the data plane with anteater," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 290–301, 2011.

[20] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey, "Veriflow: verifying network-wide invariants in real time," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 467–472, 2012.

[21] M. Canini, D. Venzano, P. Peresini, D. Kostic, J. Rexford, *et al.*, "A NICE Way to Test OpenFlow Applications.," in *NSDI*, vol. 12, pp. 127–140, 2012.

[22] P. Kazemian, G. Varghese, and N. McKeown, "Header Space Analysis: Static Checking for Networks.," in *NSDI*, pp. 113–126, 2012.

[23] T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky, "Vericon: Towards verifying controller programs in software-defined networks," in *ACM SIGPLAN Notices*, vol. 49, pp. 282–293, ACM, 2014.

[24] P. Porras, S. Cheung, M. Fong, K. Skinner, and V. Yegneswaran, "Securing the Software-Defined Network Control Layer," in *Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS), San Diego, California*, 2015.

[25] N. Paladi, C. Gehrmann, M. Aslam, and F. Morenius, "Trusted Launch of Virtual Machine Instances in Public IaaS Environments," in *Information Security and Cryptology – ICISC 2012* (T. Kwon, M.-K. Lee, and D. Kwon, eds.), vol. 7839 of *Lecture Notes in Computer Science*, pp. 309–323, Springer Berlin Heidelberg, 2013.

[26] N. Paladi, A. Michalas, and C. Gehrmann, "Domain based storage protection with secure access control for the cloud," in *Proceedings of the 2014 International Workshop on Security in Cloud Computing*, ASIACCS '14, (New York, NY, USA), ACM, 2014.

[27] N. Paladi and A. Michalas, ""One of our hosts in another country": Challenges of data geolocation in cloud storage," in *Wireless Communications, Vehicular Technology, Information Theory and Aerospace Electronic Systems (VITAE), 2014 4th International Conference on*, pp. 1–6, May 2014.

[28] N. Paladi, M. Aslam, and C. Gehrmann, "Trusted geolocation-aware data placement in infrastructure clouds," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*, pp. 352–360, IEEE, 2014.